

Introduction to Stata

ARE C253/ PP253 Fall 2003

Copyright ©2003 James Manley All Rights Reserved.

version 3: 9/8/03

(now describing some differences between Stata 7 & Stata 8, including the **graph** command)

This handout is designed to give you an idea of how things work in Stata: it was NOT written to do any particular assignment. The sample commands you type through the course of this tutorial were not chosen because the particular operation they perform is useful: once we get started manipulating the data, the sample commands mostly just generate a bunch of clutter that you won't want to save, so don't bother saving the modified dataset unless you have some specific reason for doing so.

Before we get started, though, what *is* Stata? Stata is a software package designed to manipulate and analyze data. (Some similar programs are SAS, SPSS, and Shazam. Excel also has a bit of statistical analysis functionality, but its emphasis is different.)

ORGANIZATION

Let's set up some workspace. I recommend that you create three directories in an easily accessible place. For myself, working with Windows, I opened My Computer and then opened my C: directory. There I created a directory called PP253. Within that I created three directories: data, do, and logs.

(You can do this wherever you want, or you can follow my example. It will be easier if you avoid directories like "My Documents" that have spaces in them, since Stata's not smart enough to deal with the more recent innovations in file naming. The three directories data, do, and logs are useful organizationally, but not strictly necessary.)

UNIX

If you're using the Evans 616 lab, you will need to know a few commands to manipulate that environment. The next few commands are to be typed into the command windows that pop up right after you log in. These windows are called "X-term" windows, and you can make another one whenever you need it by right clicking (and holding) on the bluish background area, selecting programs, and then selecting X-term. To close an X-term window (or any other window) you should right click on the gray bar at the top of the window, again hold, and click Quit.

You can use the command **mkdir** to create directories. Example: **mkdir logs** creates a directory called logs. To change into another directory, use **cd**, so for example to go into the newly created **logs** directory type **cd logs**. When you're logged into these machines, the data is available at */class/data/a253/[yourlogin]/data* where *[yourlogin]* looks something like a253ai. This data will not be something you can modify, so if you want to save a modified dataset, give it a different name and save it in your own home directory. ***You only have 15 megabytes of storage space not including the original dataset, which is in shared space. Saving a modified dataset is almost guaranteed to take at least 5 megabytes, so be careful and don't try to save too many versions of the data! You really shouldn't need to save any, or one at the most, if your program does everything it's supposed to.

I don't think you should need many other Unix commands, but in case you do, the system has help files - at the prompt, type **help- -l** to get the list of help filenames, and then type **help- [file]** to read it. There is also a Unix tutorial at elsa/eml/emltutorials.html if you feel the need. To run Stata in a Unix environment, type **xstata**.

To access the machines in Evans, you have two alternatives. One option is to go to there yourself and type directly into the machine of your choice. You can do this anytime: the lab is open 24/7. (Remember, the code to get in the door is 1-3, 5, 2.) The one catch is that the outside doors to the building aren't open all the time: unless you have a friend with a key to Evans, you'll only be able to get in until 7:30 pm on weekdays. There are some weekend hours, but I'm not sure of the schedule. Don't count on getting in on a Sunday.

Fortunately, the other way to get into the machines is always accessible. Any computer with internet access, Exceed, and SSH [Secure Shell] can remotely log in to these machines. (These programs are available on the Connected@Berkeley CD available through the Scholar's Workstation on campus. I think it's even legal for me to encourage you to steal the software from friends, since I think the University has a site license.) If you're having trouble getting **xstata** to run, go into "Edit Profiles" on SSH and make sure the box is checked marked "X11 Tunneling" for the profile you're using to log on.

To log on, use the machine name *emilyX.berkeley.edu* where X is a number between 1 and 15. (for example, use *emily6.berkeley.edu*) You should have a login name and password. What's going on here is that you're actually connecting to a machine in Evans remotely, even though someone else may be working on that machine: the computers can handle many people simultaneously. Because of this, when you're working directly on the Evans computers, NEVER POWER OFF your machine! If you do, you could destroy someone else's work. Anyway, back to Stata.

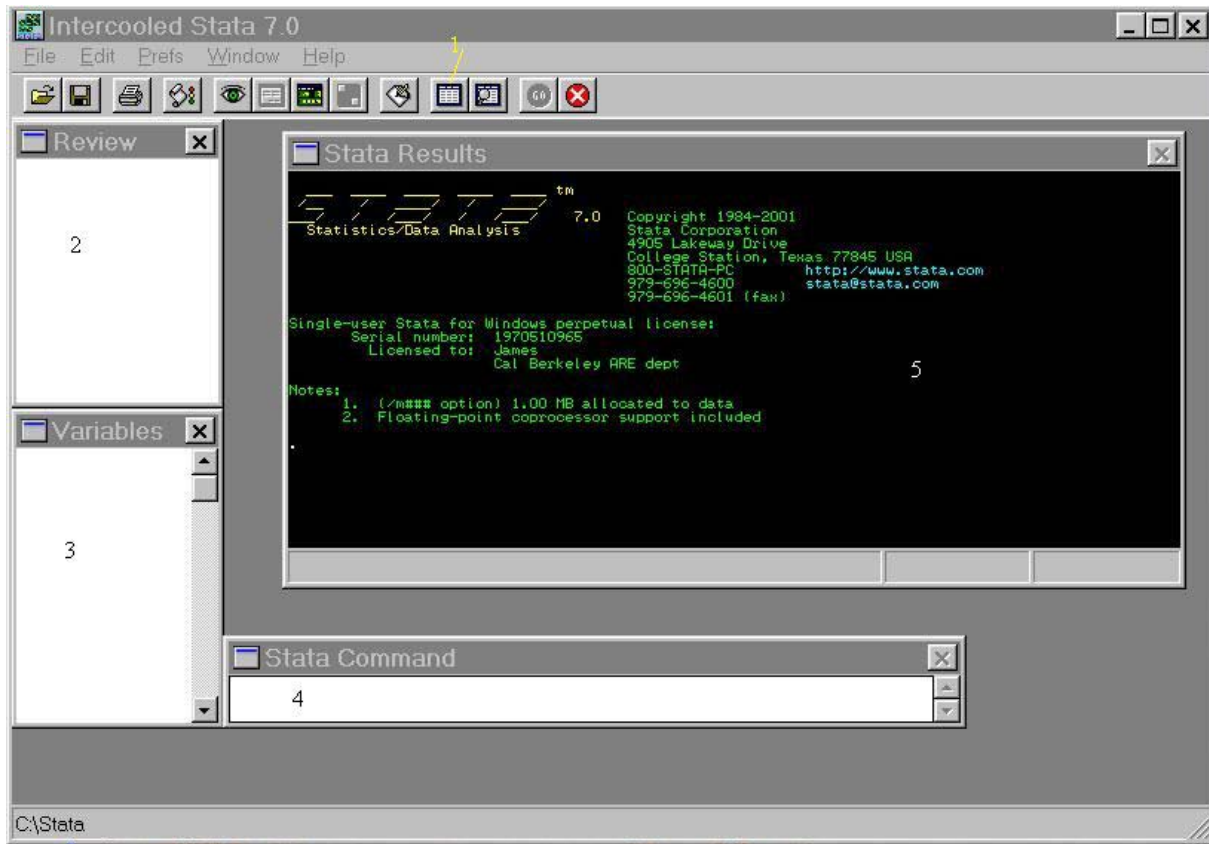
Part I: Checking out the data

LOADING DATA

You can't do much without any data, so let's load your dataset. In order to do this on your home computer, you need to have saved the data from the internet into a directory you have access to [unless you're on a Unix machine and already have direct access]. The data can be found on the class website (<http://are.berkeley.edu/courses/ARE253/2003/index.html>). The file is called *hhpov6*; put it in your new "data" directory.

If you're using (or logged into) the Evans 616 lab, the data's accessible to your account already. Click on the "open" folder, then make sure the directory says */class/a253/[yourlogin]*. Double-click on *data* and then open the file by double-clicking on it. Once you have saved the data in a convenient place, we can take Stata for a quick spin to see how things work. Open Stata by clicking on the icon in a Windows environment or by typing **xstata** in a Unix environment.

When you start the program, the screen will look something like this (without the numbers 1-5, of course) [Also, screen shots may differ from what you see in areas like the relative sizes of the windows, etc. Don't worry about that!]



A few familiar icons are in the upper left for opening or saving files. Here they're for opening or saving datasets rather than programs. The other icon I use a lot is labeled "1". It displays the dataset. (Advanced users may also make use of the last icon, which is a "break" key.)

The four windows are the heart of the interface. In the upper left hand corner labeled "2" is the Review window, which will list all the commands you type in or otherwise execute, even the mistakes. The Variables window, number "3", lists the variables in the dataset. It lists them in the order they appear in the dataset. Window "4" is the command window, which is where you can tell Stata what to do. Window "5" is what Stata uses to respond to you: it's the output window.

To open a data file on a Windows computer, we just click on the "opening" file folder in the upper left corner of the screen and move to the directory where you saved the data file. (Alternatively just type **use c:/pp253/data/hhpov6** in the Stata Command box, #4 in the above diagram). Here's what happened when I tried that:

```
Stata Results
-----
Stata 7.0
Copyright 1984-2001
Stata Corporation
4905 Lakeway Drive
College Station, Texas 77845 USA
800-STATA-PC      http://www.stata.com
979-696-4600     stata@stata.com
979-696-4601 (fax)

Single-user Stata for Windows perpetual license:
Serial number: 1970510965
Licensed to: James
             Cal Berkeley ARE dept

Notes:
1. (/m### option) 1.00 MB allocated to data
2. Floating-point coprocessor support included

. use "C:\PP253\data\hhpov6.dta", clear
no room to add more observations
r(901);
```

Oops! An error ALREADY. [Apparently Stata 8 deals with this better than my poor version 7, but bear with me in case you need these commands.] The data file we have for this assignment is too big! Stata starts off allocating a certain amount of space for data, and the fat dataset we have is too much for it. We need to increase that allocation by typing **set memory 10m** in the Stata Command box (which is where your cursor will be automatically). This command allocates 10 megabytes of memory for the data, which is more than enough for this dataset.


Note that the “use” command, whether it was typed or selected via mouse, appeared in the Review window after it was executed. This is a convenient time to note the usefulness of this window. If you push the Page Up key, the last command you typed (the last one in the Review window) will appear in your Stata Command box. Pushing your Enter key will execute the command again. So instead of going through all the clicking or typing again, just hit Page Up twice (until the “use” command appears in the Stata Command box) and then hit Enter. This time, the data should load successfully. (You can also pull commands out of the Review window by clicking on them rather than using Page Up/ Page Down. Retyping the command or going through the mouse-based selection process will still work, of course.)

Another problem that occasionally crops up is a limitation on the number of variables. Stata version 7 is initialized to deal with only 40 variables. Obviously this wasn’t a problem for this problem set, but it can be with other datasets or if we start creating a lot of new variables. The way around this is to use the command **set matsize XX**, where XX is a number up to 800.

A last *set* command is **set more on** and its counterpart **set more off**. When you execute a command that requires Stata to generate more output than can fit in the window without scrolling, normally it fills the screen and then writes **more** at the bottom. When you press the space bar it gives you the rest of the output. (Hit **q** if you don’t need to see any more.) If you want to turn this option off so output will just stream through the window, just use **set more off**. Then Stata will simply spew all of the output to the screen directly without waiting for you to

read it. This may not seem useful now, but when you're capturing output using a log, you don't want to have to hit the spacebar dozens of times to get through the program!

THE DATA EDITOR

Sometimes it just helps to get a look at the dataset, huge though it may be. The button  lets you look through all the observations. While the Stata Editor window is open, no commands can be entered: the Stata Command window disappears completely. Also, while this window is open you can change the data, so be careful! Finally, data appearing in this window may be labels rather than actual values. Use the arrow keys to move the cursor one box to the right. Part of the display should show

```
region[1] = 1
```

region	depto
metr	guat

Note that although the highlighted portion says "metr", the above indicator region[1] lets you know that the actual value of that variable for that observation is the number 1, not letters at all. We'll discuss this more later when we cover labeling.

On this screen, you can sort the data by any variable. If you want the observations in order by their segmentos, just click on *segmento* and then click on *sort*. You can also hide variables and delete observations, variables, or individual values. Sorting can be done outside of this screen using the **sort** command. For example, **sort region depto mupio** sorts the data first by region, then within that by departamento and then by municipality.

When you're done in here, just close the window, and your Stata Command box will reappear. If you've sorted the data or altered it in any way, it will ask you if you want to preserve what you've done or if you want to restore it to the way it was before you messed with it. [To close a window on the Evans 616 Unix machines, right click on it (you may have to hold it for a sec) and then choose Close.]

VARIABLES & DATA INFORMATION

Now you can see that the Variables window is full of words, some of which are apparently gibberish. Scroll down to see them all. (Yes, really do this: it's a good idea to have some idea what kind of variables you have accessible. My personal favorite is *pexclsew*- what's yours?) Here're a couple of ways how to figure out what these creatively named variables represent:

describe lists all the variables along with their interpretations. Just type **describe** in the Stata Command box and you'll get a listing of all the variables with their labels. If you don't need to see all the variables, just list the names of those variables you want to learn about: try typing **describe drenaje sewnet**. [There's also an abbreviation for describe: you can type **desc** or just **d**.]

tab (short for tabulate) tells you how many occurrences of a certain variable are in the dataset. For some variables, like *urban*, this is handy. Type in **tab urban**. You should see something like this:

```

. tab urban
urban area
(rural
excluded)

```

	Freq.	Percent	Cum.
0	3852	52.94	52.94
1	3424	47.06	100.00
Total	7276	100.00	

There are many useful pieces of information here. First, in the upper left hand corner, it tells you what the variable represents. Below that are the two values the variable takes in the sample, and to the right of those are the number of times those values appear in the dataset. In this case, 3852 of 7276 observations or 52.94% take the value of 0, which means that those households are rural rather than urban.

An additional bit of functionality **tab** has is to generate dummy variables based on tabbed output. for example, if we type **tab region, gen(reg)** Stata first gives you the output and then generates a dummy variable for each region, with the variables taking the names *reg1* – *reg8* in the order they were in the original dataset. Variable *reg1* thus takes the value 1 for observations located in the region that had the lowest numerical value (which in this case is called “metropol”) and is zero for observations in all other regions. In Stata 8 there’s a nice bonus- all the new variables you’ve created have labels already, so you know what they are. Type **desc reg5** to see.

For other variables, **tab** is less than useful. Try **tab hogar**- since each household has a unique value, there are over 7000 values, and Stata knows you don’t want to go through 7000 values, so it just refuses to show it to you. [I should say it *did*, in Stata 7; in 8 you get to page through them all. Just hit **q** to get out!] Try **tab sector**- it’s not quite as bad as the previous example, and Stata actually executes the command, but you have to hit <space> a lot of times to get through all the possible values. (Hit **q** when it says –more– to give up partway through.)

Finally, **tab** has yet another purpose: it can compare two variables. Try **tab urban indigena**. It puts the “urban” variable on the vertical axis and the “indigenous” variable on the horizontal axis, and lists how many observations fall into each grouping. Here of 3424 urban households, 879 are indigenous.

```

. tab urban indigena

```

urban area (rural excluded)	indigenous household		Total
	0	1	
0	2075	1777	3852
1	2545	879	3424
Total	4620	2656	7276

To get percentages, try adding the word “row” after a comma: ex. **tab region urban, row**.

```
. tab region urban, row
```

region politico administra tiva	urban area (rural excluded)		Total
	0	1.289063	
metropol	119 12.85	807 87.15	926 100.00
norte	406 50.88	392 49.12	798 100.00
nororien	224 37.40	375 62.60	599 100.00
surorien	479 59.50	326 40.50	805 100.00
central	824 65.87	427 34.13	1251 100.00
suroccid	688 61.70	427 38.30	1115 100.00
noroccid	793 66.25	404 33.75	1197 100.00
peten	319 54.53	266 45.47	585 100.00
Total	3852 52.94	3424 47.06	7276 100.00

If you're a graphical person you can experiment with the **graph** command. This has changed substantially in version 8, so much so that my old commands don't work. Fortunately, they left a little back door for us. The **graph7** command tells Stata 8 to pretend it's Stata 7 and graph accordingly. Since that's what I'm familiar with I'm describing those features, but you're encouraged to explore Stata 8's new functionality. In Stata 8 type **help twoway** or **help scatter**, and of course **help graph** also has good information. To test the old features, try **graph7 telefono** and **graph7 factor2 factor3**. Now try **graph7 hogar**. Remember that when we typed **tab hogar** Stata 7 balked, saying that there were too many values? But now it looks like there are only five values! Stata's graph function automatically renders data into a histogram, grouping the observations. This brings us to:

***** **HELP** *****

If you're not sure about what a command is doing or about the syntax you should use with a command, just type **help** and the command. For example, **help graph** will tell you about using the **graph** command.

To open a separate window for your help screen, go through the pull down menu marked *Help* and choose *Stata Command*. Then type the command you're looking for. If you're not sure of the exact command name, use the *Search* menu option. You often have to dig through the search results to get what you want using this approach, but at least it's something.

Another very accessible resource is the huge internet community of Stata users. There're countless web pages dedicated to explaining all types of Stata commands, sharing programs, etc., and I've learned a lot online myself. If you get stuck, just throw words like *Stata label define* into Google and see what comes out. A lot of times it's more helpful than even the online manual, though as you might expect with the internet, you usually have to root around a lot to find the good stuff.

SUMMARY STATISTICS

summarize [variables] gives sample statistics for a given set of variables. Type **summarize segmento upm2 drenaje**. Stata tells you the mean, standard deviation, maximum, and minimum of the values the three variables take. [The abbreviation for the summarize command is **sum**, so you can just type **sum segmento upm2** to see the information about those two variables.]

To get information on percentiles, skewness, and kurtosis, you can use summarize plus **, detail** : **summarize segmento upm2 drenaje, detail**. Don't forget the comma. I hope that you know more about skewness and kurtosis than I do!

To get summary statistics only for households with heads working between 12 & 20 hours (i.e. variable *horal*) we use "weights" to restrict the sample. To find out how many of these households have heads working in the mining sector, type **sum work1 [w=horal>0]**. (Note that we're using square brackets and not parentheses.) Weights are more complicated than I can describe quickly; if you really want to learn more, type **help weights**, but it's ugly: I don't recommend it (and you won't need more than the above for this!). The bottom line is that you're using the part of the command in the brackets to tell Stata that you only want to consider part of the population (in this case, households with a value greater than zero in the "horal" column. (Forgot what *horal* is? Use **describe** to check!) Then Stata applies the first half of the command, i.e. **sum work1**, to just that group of people (in this case, it's just over 200 households.)

TABLES

To output data in a table format, use the **table** command. The first variable or variables to be listed are those in the rows. The columns are next but note that each variable in the columns requires a command: easy ones include *mean*, *sum*, and for number of observations with real values (i.e. not missing values) use *count*. Example: **table region, c(count work1 mean work1 sum work1)**. Try it!

Part II: Manipulating data

Ok, you know the data's there and you have some idea about what it is. Now comes the real work: messing with the data. You have a bunch of variables in your dataset already, but you will need to modify some of them and create others.

To create a new variable, use the command **generate** (or one of its abbreviations, **gen** or just **g**). For example, to create a variable for total annual income squared, type **gen income_2 =ingtotpc*ingtotpc** (note that you can put spaces around the equal sign or not- it's up to you)

Generate also works with selecting minima, for example, **gen lowwork=min(work1, work2)** will create the variable *lowwork* where each value is the minimum of the two variables in parentheses (in this case, all zeros).

A couple of other values you may wish to get at for some reason are **_n**, which represents the number of each observation, and **_N**, which represents the total number of observations. You can use the former with the **sort** command to get rankings: for example, first sort the data by per

capita consumption and then generate a new variable numbering them by typing **sort conspc** and then
gen incomerank=_n.

DEALING WITH DUMMIES

The generate command also handles “dummy” variables (i.e. variables that are just 1 or 0, like *urban*). For example, **gen powrphon = telefono & electric** creates a dummy variable that is 1 only if a household has both electricity and a phone. **gen powORphon = telefono | electric**, on the other hand, creates a variable that has the value 1 if a household has either a phone, electricity, or both. [The | character is above the backslash character \ on a key just above the Enter key on most keyboards.] To create a dummy variable that takes the value of 1 if a household has 3 or more children under the age of 7 (and 0 otherwise), type **gen lotsakids = n0_6 > 2** (you could also use **>=3**). Note that 1 and 0 are basically acting like true and false here, so **4<=9** generates a 1 for true.

***Potential problem: Stata uses the character = as an assignment operator. Whatever’s on the left is being assigned the value on the right. If you want to see if two things are equal, then, you must use two equal signs in a row. **gen sevenkids = n0_6 ==7** generates a 1 if the household has seven children between the ages of 0 and 6, and a 0 otherwise.

FUNCTIONS

Annoyingly, Stata also has another command for generating variables, and you have to use the right one or it gets mad at you. Use the command **egen** to create new variables using functions. Functions are anything like *mean*, *sum*, *median*, etc. For example, if it was important to know how many people lived in the “peten” region, you could type **egen petenfolk=sum(region8)**. A variable would be created with every observation having the value 585. This may not seem useful, but you’ll probably need it!

An interesting list to browse is the list of functions: just type **help functions** and Stata will spew out a long list of all the functions. Also, the egen command is pretty powerful; to see all of the things it does just type **help egen** and scroll through. Let’s take a look at just one more, the egen “pctile” command. To identify the fiftieth percentile in terms of group membership (i.e. 50% of the sample participates in this many or fewer groups) type **egen halfclubs=pctile(numorcl), p(50)**. This is like asking Stata the question: “If I wanted to draw a line, based on the number of groups people participate in, below which half the households would fall, where would I draw the line?” The way Stata responds to your question is by creating a variable and filling it in with its answer, not just once but once for every household, so the same number will appear 7276 times, in this case. This may seem unnecessary, but it can be convenient. It’s very easy to compare all households against this standard now that every household has it.

To get the 60th percentile, just change the 50 to a 60. (What’s another name for the value at the 50th percentile, i.e. the value at which 50% of the population is below the line?) You can also pick this out manually using **tab** and the cumulative distribution listing, but this is quicker.

RECODING & RENAMING

Sometimes it may be useful to rename a variable or to change some of the values. If, say, the variable *ed_1_5x* isn't the easiest for you to remember, you can call it something else. Since the variable takes a 1 if the most educated member of the household hasn't finished primary school (I assume!), we might want to rename the variable: **rename ed_1_5x priminc**. This isn't much better, I admit, and I encourage you to choose your own variable names. Descriptive names are always the most useful. Stata will let you call a variable anything you like, and it does keep track of upper/lower case, so make sure you do too. Generally I find it easier to avoid using upper case for variable names.

There are also times when it's useful to convert values. For example, when I first downloaded the dataset, the 1's for one of the dummy variables had been changed to 1.2896025! I really didn't want to go into the data editor and change them all one by one, so I used the **recode** command. To recode the 1's for the dummy variable *pexchuma* into 75's, just type **recode pexchuma 1=75**.

Finally, you can recode conditionally using the **replace** command. For example, to replace values of one variable (*particip*) based on values of another (*pbridgin*), use **replace particip=1 if pbridgin>=1**

The conditionality need not be dependent on another variable: it can be on the same variable, for example if you want to replace all consumption expenditures equal to or higher than 50 qq with the value 999, type **replace conspc=999 if conspc>=50**.

DROP & KEEP

One last tidbit in this section: to cut down the number of variables in your dataset, you can use either of two commands: **drop** or **keep**. As you might expect, **drop** deletes all variables you list afterwards, so **drop clinking pexcedu** drops those two variables. **keep** deletes everything but what you list, so **keep work7 work8 work9** deletes all but these three variables. You can also use * as a wildcard: to drop the variables *hora1 hora2 hora3 hora1_2 hora2_2* and *hora 3_2* you can just type **drop hora***.

SAVING DATA


After you're through manipulating your dataset, always remember to save it. The command is simply **save filename**. I recommend that you be explicit about where you want it to go: **save C:\PP253\data\sampldat**. I also recommend that you not save over the original data that you've downloaded. Use a new filename for data you've altered and keep the original as a backup. (Of course, you can always download it again if you need it, but it's good to get in practice for when you have the only copy of a dataset!)

There's another temporary way to save data. Typing **preserve** by itself saves a copy of the data as it is in Stata's memory. Then you can alter it, keep or drop variables or whatever, and when you want it back the way it was, just type **restore**. This is a handy trick you'll use a lot.

Part III: Programming

You can do a lot just by typing in a series of commands, but you can do more by making a big list of commands and executing it at once. That's all a computer program is: just a list of commands like the ones we've been working with. A program in Stata is known as a "do-file" because when it's saved it has the file extension `.do`. You can use any text editor to create programs, save them as text files with a `.do` extension, and Stata will happily run them. In Windows, you'll just be able to click on the program you've written and it'll be run through Stata. Stata also has a built in editor for writing programs, and this explanation will work with that interface.

I recommend you start using this window immediately and don't look back. Click on Windows and find the Do-File Editor. A new window will open up, which works just the same as the command window except that you can save everything you type in here, and nothing executes

until you push the  button (inverted, sorry), at which point all the commands you have listed there will run (unless you have one highlighted, in which case only that one command will run). This way when you get a few commands that work well strung together, you won't have to go through the whole process of typing each one in again when you want it. Make sure to save your "do-file" regularly.

LOGS

To use programs most effectively, you need to save the output they generate. Actually you can do this in an interactive (i.e. non-programming) context as well, but it becomes especially important here. A set of saved output is called a *log*, and not surprisingly you start one using the command `log`. You need to be a bit more explicit than that, however: specifically, you need to tell Stata where to save the log file. The full syntax is `log using filename`. Again, I recommend you be explicit about the path you want the log saved in. Note that Stata won't overwrite an existing file unless it's explicitly told to do so, which means that the second time you run this program with the same log command, you'll hit a snag. Get around this by adding `, replace` to the command, telling Stata to go ahead and overwrite the file. If it's the first time the file is being created, Stata will give a little warning but the program will continue; if you don't include the replace label and it's needed, the program will just stop. Stata can be stubborn. Anyway, the full command should look like `log using mylog, replace`.

One other issue crops up with respect to logs. Most machines are set to write logs in Stata's own special format, which has a `.smcl` extension. If you create a log in this format you can access it using the View option in Stata's File menu. Apparently it has some useful characteristics, but I find it easier to have Stata write logs that are simply text files. The command to change the logs to text format is

`set logtype text`. Make sure you put that command before you start the log!

At the end of the file, make sure to close your log. The command for this is just `log close`. Stata will let you know where it's saved the log.

EDITING PROGRAMS

To open Stata's "Do-file editor," just go into the Window menu at the top and choose the appropriate option. On a Windows machine you can call it up using Ctrl-8. The buttons at the top

are mostly familiar, with the exception of the last two. The empty white box with the arrow down runs the file you're currently looking at: it does everything at once, so if there are any errors, you just get an error and nothing happens. The white box with lines and an arrow executes the file line by line, command by command. This is what I usually use.

An absolute MUST when programming is the use of comments. Leave remarks in the text of the program so that when you look at it later you know what the program does. Always start off a program by listing the approximate date, purpose of the program, and who wrote it. Then when you finish modifying it put something here at the beginning of the program so that you know what the last modifications were. To enter comments in Stata, just start a line with a * (asterisk). Comments longer than one line can be denoted by starting with /* and ending with */. For example, you could just enter this entire paragraph, and then set these marks at the beginning and end of the text. Stata will happily ignore it, not trying to enter any part of it as a command. In addition to one big comment at the top, you should add comments through your program where necessary to explain what's happening in a given set of commands.

There's one other use for these comment markers. Since what they do is to tell Stata to ignore everything between them, you can use them to write commands longer than one line and have Stata read it in as one line. Example:

```
summarize sewnet area tuberia region2 region3 /*  
        */ region4 region5
```

will be read in by Stata as

```
summarize sewnet area tuberia region2 region3 region4 region5
```

After you write your initial comment, just start adding commands. Load the data, run your processes, etc. You can display output directly to the screen using the **display** command.

Using the editor is pretty easy. There are menus to do what you need. When you save files, I recommend you use the *do* directory you created earlier. Don't forget to make sure that the files you save have the .do suffix.

DELIMITERS

One option you can play with is the use of the #delimit command. The default option is for each command to be entered on its own line. Stata will then accept a carriage return as the end of a command. For example, you could type

```
set mem 10m  
use c:\PP253\data\hhpov6  
tab urban region, row
```

[Note that **use** is the command for loading data if you're typing rather than using a mouse- you'll need it when programming.]

Some programmers prefer to be able to write multiple commands on a single line. For this, use the command **#delimit;** which sets the semicolon as the “end of command” flag for Stata. Then commands need not have their own line; you could type

```
#delimit;  
set more off; set logtype text; log using c:\PP253\logs\firsttry; tab urban region, row;
```

and all five commands would execute.

LABELLING DATA

The commands like **describe** and **tab** list information about variables only if it's been programmed in by someone. Try creating a new variable using **gen** and then use **tab** or **describe** on it. Since the new variable hasn't been labeled yet, Stata doesn't tell you much about it. To attach a label to a variable, use the command **label** with the specific command **var**. For example,

```
tab region, gen(reg)  
gen urbanmet=urban*reg1  
label var urbanmet “Household is in urban part of metropol region”
```

When you type **tab depto**, Stata lists the values the variable *depto* takes and how many times each value is found in the dataset. Notice that the values, though, are words and not numbers. Before, when we looked at the data editor, we saw that what was stored in Stata was actually a name. To associate names with these numbers, use **label define** coupled with **label values**. For example,

```
label define UMLabels 1 “Urban&Metropol” 0 “NotU&M”
```

creates a set of labels called *UMLabels*. To attach these to the values of the variable we just created, type

```
label values urbanmet UMLabels
```

Now try **tab urbanmet** to see that the value of 1 is now labeled “Urban&Metropol” and the value of 0 is now called “NotU&M.”

REPEATING COMMANDS

To repeat a command several times, you can use the **for** command. (Stata offers many options including **foreach**, **while**, **forvalues**, and **if**. Please see the help files for the syntax.)

The **for** command repeats things as many times as you give it arguments. Each argument is taken once through the command, substituted in where the X is, as follows:

```
for var ed_1_5x ed_6x ed_7_10x : replace X 1=35
```

In this example, all the 1's in each of the three dummy variables is replaced by the number 35.

Another way to repeat something is to use a macro. If you are going to be using a given string of variables (or any set of words, actually) many times, you can create a macro and just type the macro wherever you want those words to appear. To create a macro, use the **global** command with the name of the macro you want to create. To invoke it, use the **\$**. For example,

```
global agegrps "n0_6 n7_24 n25_59 n60_plus"  
summarize $agegrps  
for var $agegrps: gen Xa=X*urb
```

[Note that *urb* here works just as well as typing the whole variable name *urban* would. As long as there's no doubt about which variable you mean, i.e. only one variable that begins *urb*, Stata will assume that's the one you mean.]

prints out the summary information for each variable and then creates four new variables, each named just like the four in quotation marks but with the letters "urb" added, which are the initial four variables multiplied by the dummy variable *urban*. For example, the new variable *n0_6urb* is the variable *n0_6 * urban*.

MERGING DATASETS

To unite datasets, use the **merge** command. We don't have another one to merge into this one at the moment, but if you save several datasets you can later join them using this command.

Syntactically, type **merge variables using dataset**. Before you merge, you must ALWAYS **sort** by the same variables that you're about to merge by. In addition, the data you are loading must also have been so sorted. You can never assume that the data have been sorted: if you're merging data files later the last thing you do before you save a dataset must be to sort it.

Part IV: Data Analysis

As you know, one of the main reasons the professors of this class wanted to train you on Stata was so that you could do a *probit* model. Ordinary least squares regressions should not be used with dummy variables as the dependent variable for several reasons, including the fact that the estimated value of the dependent variable might be able to exceed the 0-1 range for some set of parameters. This obviously wouldn't make a very good prediction! A *probit* model uses a modified normal distribution to generate for the dependent variable a set of probabilities which cannot exceed 1 or go below 0. Since it's based on the normal distribution it's a good choice for economic data, particularly when we have lots of observations.

To run a probit model you just type the command **probit** followed by the dependent variable and then the independent (explanatory) variables. For example, if you had a new theory by which ethnicity, number of young children, and some other variables affected a household's being on the local sewer network,

```
probit sewnet n0_6 hhhfemal espanol indigena por_male
```

would regress the variable *sewnet* on the other listed variables, telling you whether the explanatory variables are statistically significant predictors of the *sewnet*. The reported coefficients represent the level of correlation between that variable and the dependent variable.

One other piece of information which might be useful is the level of contribution of each dependent variable to the independent variable. For example, say that having a female household head makes a household 50% more likely to be impoverished, while being a Spanish-speaking household decreases the probability that a household is poor by 30%. The first probit regression won't give you this information, but you can get those "contributions" by using the **dprobit** command. Just type the command as before:

```
dprobit sewnet n0_6 hhhfemal espanol indigena por_male
```

T-TESTS

This one is tough: to do a t-test of a variable mean for statistical significance, type the name of the variable you wish to test and then the value you want to compare the variable to. For example,

```
ttest hh_age=45
```

CORRELATION

To get a table of correlations, just type **correlate** and a list of variables. You can also get covariance:

```
correlate n0_6 urban work6  
correlate n0_6 urban work6, covariance
```

(and no, there's no reason to expect these to be correlated!)

SURVEY WEIGHTS

Another aspect of Stata that's very convenient with data like these is its series of "survey" commands. Stata knows how to deal with survey data: you can set up strata within which to do analyses and set weights Stata will use to transform stratified data such that the statistics it generates are representative of the whole population. For the first assignment, the professors have generously allowed you to ignore them, but who knows if they will again? Good to know, anyway.

To use this level of functionality, you first have to give Stata some information. First, you have to know how the sample is stratified. Next, you need to know the "primary sampling unit." Finally, you have to choose the weights. You can tell Stata about each of these using the **svyset** command with an appropriate sub-command:

```
svyset strata stratname
```

```
svyset psu identname
```

```
svyset pweight weightname
```

where *stratname* is the variable that identifies the type of stratification, *identname* is the variable that identifies the sampling unit, and *weightname* identifies the expansion factor. All of these are written into the text of your assignment: refer to that for the variable names.

***NOTE: the above syntax is correct for Stata 7.0, but has been altered for version 8.0. In 8.0, it's something like

```
svyset, strata (stratname)  
svyset, psu (identname)  
svyset, pweight (weightname)
```

After you've set these parameters, you can use the functions directly: **svytab** works like regular *tab* above, **svymean** gives means for the strata, **svyratio a b** gives the ratio of a/b (hint: use this and totals to generate shares for continuous variables!), and **svyprobit** runs probits. Each of these commands can be applied to different groups within the sample through use of the **, by** suffix. For example:

```
svymean n0_6 n7_24 n60_plus, by (urban)
```

generates the mean number of people in each age group per household in urban and non-urban contexts.

Finally, if you need to compare some of these means, you can use the **svylc** command. As the manual puts it, "Estimating differences in subpopulations means, for example, can be done by running **svymean** with a **by()** option, and then running **svylc**. This is the equivalent of a two-sample t test for survey data." For example, after the above **svymean** command, you could put

```
svylc [n0_6]0 – [n0_6]1
```

Since the **svymean** command separated the sample into groups by whether the household was urban or rural, we can access the separated groups using the brackets with the 0 and 1 after them. This command looks at the **n0_6** values of the first group, and takes their mean (because that's what the previous command did) and then subtracts the mean of the second group. Finally, it runs a test to see if the result is significantly different from zero.

That's it for now: you should be able to do most, or even all of the assignment using these commands. When in doubt, reference the built-in help, as it's very complete and usually includes examples of command syntax. Stata is a very powerful package with a lot of functionality not offered by other programs. It's in increasing use all over the world, and notably at the World Bank and other research institutions. Hopefully this tutorial will help you get on your way to building fluency with the package. Good luck!

***Always make sure to log out of any machine you use. To log out of the machines in Evans 616, right click on the blue background area and then select *Exit*. DO NOT TURN THE MACHINES OFF!